APPENDIX

Overview

The House Control System (HCS) is composed of several subsystems, such as HVAC, Lighting, HVAC, and Audio/Video. Each subsystem is responsible for managing its own resources and providing services to the end users of the HCS system.

The various HCS AV subsystem components provides the user complete and uniform access to the various AV resources available to HCS.

The Physical Environment

Before we dive into the details, let's take a look at the physical AV environment that HCS exists within, manage, and represent to the user.

In general, the physical AV system should be viewed as a set of "networked" devices that can be dynamically interconnected and programatically controlled.

The "AV Network"

The AV Network consists of one or more AV analog matrix switches through which the various AV devices' inputs and outputs are connected. These switches, taken together, are used to form temporary connections between these devices for the purpose of routing AV signals between them.

The AV Network could be extended to include the digital routing of AV information via high speed networks like ATM. This fact needs to be taken into account from an architectural point of view.

The AV Devices

There are several kinds of AV devices envisioned. These include:

- Various kinds of Audio Speaker systems (ambient, stereo, theater...) -- these include amplifiers,
- Various types of Video Displays,
- Remote control devices (Port-a-mice),
- Standalone recorders and players (e.g. VCR, CD...),
- Shared RF (TV & Radio) broadcast tuners,
- Other internal AV sources (such as security cameras),
- Cable "tuners" for audio and A/V broadcast (DMX, TCI...), and
- Shared pools of AV Programming:
    - Audio CD Jukeboxes,
    - LaserDisc (AV) Jukeboxes,
    - VHS Tape Jukeboxes, and
    - Still frame image libraries.

The goal of the HCS AV design is to abstract the behavior of these resources to a point that they, as well as new types of devices (hardware and software), can be dealt with in a uniform, natural fashion by the end user.

The HCS User's view

It is not appropriate for the HCS user to interact at the physical level of the AV subsystem. The goal then is to provide natural, high level, abstractions through which the user interacts. In order to accomplish this, there are several other intermediate abstractions between the physical entities (devices) and the user that will have to be defined and implemented. In order to organize our thinking as well as the design itself, a layered functionality model is used.

The HCS AV Functionality Model

The following model is used to classify the different levels of functionality within the HCS AV subsystem.

Layer	Purpose
Service	Provides end-user access and control over the entire AV subsystem. Provides natural and unified perspective of the AV resources. Components at this layer tend to be UI oriented.
Management	Provides for the management of specific kinds of resources (physical or not). The intent is to limit specific knowledge of AV resources, routing, etc. to this or lower layers. Further, the intent is that the Service layer components "use" the management layer to accumulate and carry out user requests (more about this later). It is very possible that this layer will deal directly with the user via UI component objects that are presented within Service UIs.
Network	The components of this layer are responsible for managing and representing the temporal connections between AV components involved in the execution of a user request. This layer also includes thel components implement the actual AV "switching fabric". The entire fabric, no matter how complicated, is "glued" together and presented as a single, sparse, 2-D (inputs vs. outputs) switch matrix. In addition, this layer models the AV inter-connection map of the entire AV system (circuits, ports...).

Each of the various AV objects, that make up the AV software implementation, each map there behavior to one of these levels. Once all of these component objects classes are introduced, each level of this reference will be restated identifying which components implement that level of functionality.

Review: The HCS Spatial Model

The basic unifying, the "concrete" if you will, object class in the HCS design is the *HcsSpace*. HcsSpaces represent the actual (physical) spatial entities (areas, rooms, wings, floors, buildings, ...) and their relationship to each other. This spatial model provides the "grounding" for other objects within the HCS system design. Before going into the details of the AV object model, a bit of review may be helpful.

There is a object class called an *HcsSpatialService*. HcsSpatialServices are aggregated within HcsSpace instances. Their purpose is to extend the overall behavior of a specific HcsSpace beyond that of just being a space. As an example, if a given space is to allow the control of lights, then the lighting oriented HcsSpatialService would exist in that space. The same would be true for spatial AV behavior.

To complete this space-centric behavior description of HCS, we consider how UIs are implemented within this context. First of all, all physical points of HCS/User interaction are modeled as an *HcsUserControlPoint* object and are associated with exactly one HcsSpace at any given time. There may be multiple HcsUserControlPoints associated with a single HcsSpace however. The intent is that this relationship model the physical relationship between these entities.

For every different kind of HcsUserControlPoint there will be a specific derivation implemented. The intent is that a derivation knows exactly what it "is" and how to act that way. To make this point clear: a "TV" kind of HcsUserControlPoint acts like a TV that allows HCS control to occur within that context. While a touch panel HcsUserControlPoint would allow much more direct HCS access because its basic purpose in life is to provide for general HCS system usage.

Another important fact about HcsUserControlPoints is that they are basically, beyond their normal behavior (like being a TV), just UI-frameworks for the HcsSpatialService instances that exist in that HcsSpace and to which the subject HcsUserControlPoints belong.

So far the only real UI type of component introduced has been the HcsUserControlPoint. Again from an UI perspective, this is really just a framework for other subordinate UIs. Other objects (Resources) in the system such as HcsSpatialServices and shared AV devices in the ECs, need to be able to present their UIs ultimately within the framework of specific kinds of HcsUserControlPoints (TV, PC, TouchPanels...). This is not done directly but rather through surrogate objects called an *HcsResUserCntls* (RUC) that co-reside in the same process space as the HcsUserControlPoint through which the resource's UI will be presented. One way to think about Resources that have a UI is as a set of objects. The first being the actual resource itself, and the others being the various types of *HcsResUserCntls* (UI component) for the different types of HcsUserControlPoints through which that Resource can be used.

With this as a backdrop: the AV HcsSpatialService is called HcsEntertainmentCenter. It is in fact a derivation of the HcsSpatialService object class; and as such, directly extends the behavior of the HcsSpaces into which it is configured. There may be zero or more instances of an HcsEntertainmentCenter present within a given HcsSpace. The rest of this document, as well as the bulk of the AV design and implementation, deals with abstracting the physical AV devices and interconnects to a point that they can be represented through an HcsEntertainmentCenter to the end user.

Representing Spatial AV Devices

In addition to pools of shared AV hardware resources, there can also be normal (standalone) devices present in a space. All standalone devices (e.g. displays and media players/recorders) that are physically present in a space, are grouped together into a collection (HcsCollection) called an *HcsAvCollection*. HcsAvCollections are aggregated within their "owning" HcsSpace instance. Given this relationship, an HcsSpace can indirectly identify all of its local AV components to other HCS components. NOTE: shared devices such as JukeBoxes are NOT in this category -- they are not considered to be local devices within any space.

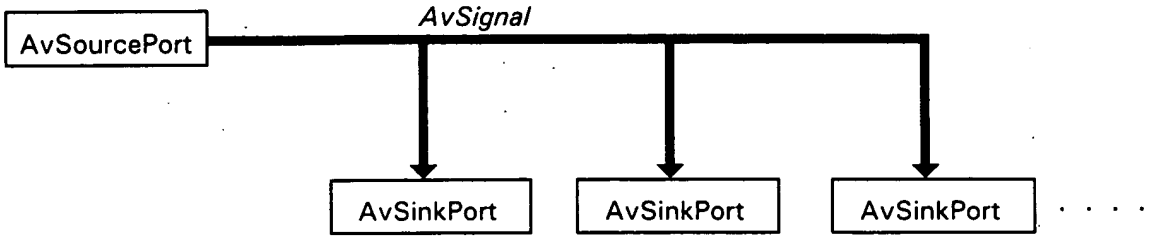
Definitions

This section of the document provides a description of the HCS AV object model through the definition of each object class within that model. Some of these object classes represent very low level kinds of things like wires and such, while others represent abstract things like information. An important point is that none of object classes defined here are just intended to model their real world (physical) counterparts but also "soft world" counterparts as well.

The best way to absorb the overall object model is to read the set of definitions at least twice. This is necessary because of the cross references between various definitions.

Audio/Video Signal

The ***AVSignal*** object class represents all actual real time audio/video information that is sourced, routed, and sinked within an HCS AV network. This object class is intended to not only model analog signals but "digital signals" as well.

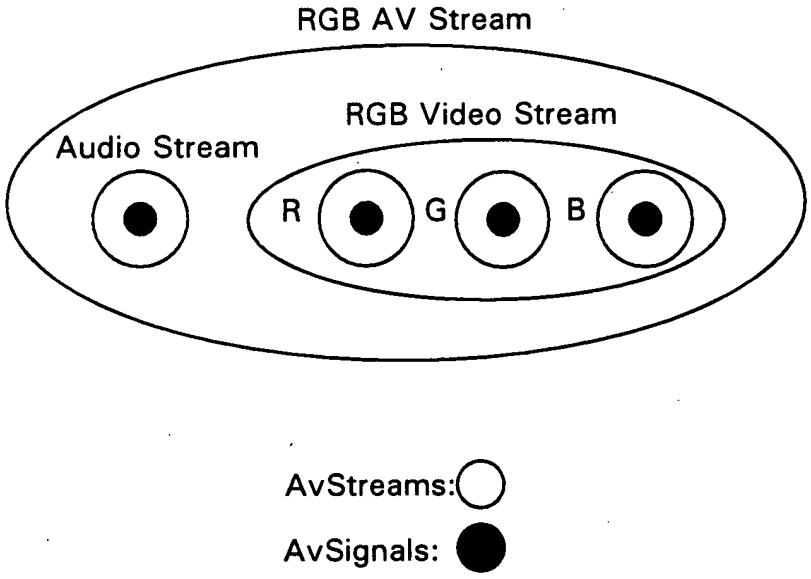


NOTE: AvSignals are directional and hierarchically classified (typed) using ASN.1 Object Ids. AvSignals are directional in that they originate at an AvSourcePort and are terminated at one or more AvSinkPorts. More about ports later.

Audio/Video Stream

The ***AVStream*** object class represents a "directed container" of either an AvSignal or subordinate AvStreams. Multiple AvStreams can be contained within a common parent AvStream only if the "sub-AvStreams" are related to each other by some constraint, such as synchronization. An AvStream is "directed" in that it originates at an AvSourcePort and is terminated at one or more AvSinkPorts. The containment relationship of an AvStream can be arbitrarily complex.

Example: an RGB Audio/Video Feed:



Definition: *Primitive* AvStream: A primitive AvStream is an AvStream that contains only an AvSignal.

Definition: *Complete* AvStream: A complete AvStream is an AvStream that is not contained by another AvStream.

In general, AvStreams are intended to contain all of the attributes needed to fully describe the AV data that the AvStream represents. This will include some, and possibly a large number of, textual attributes. This idea also extends to the representation of component signals of an alternate form. For instance, an AvStream that represents the audio and video analog outputs from a VCR: if the contained analog AvSignals were routed through an MPEG encoder, that encoder's output AvSignal(s) would also be included in the same AvStream along with their analog cousins. In other

words, there is still only one AvStream which contains all of the different representations (digital and analog), in terms of AvSignals, of the original information.

Audio/Video Source Port

The ***AvSourcePort*** object class represents the point of origin of a single AvStream and thus the AvSignals which that AvStream contains. More specifically, AvSourcePorts can represent a "physical" source of an AvSignal (primitive AvStream) or some higher level aggregation of AvSourcePorts. The aggregation hierarchy of AvSourcePorts corresponds to that of the AvStream "produced" by that AvSourcePort. Assuming the above AV RGB Feed as an example: there would be a single high level AvSourcePort that "produces" one AvStream in the form of the "RGB AV Stream". This AvSourcePort would then contain two other AvSourcePorts corresponding to the "Audio Stream" and "RGB Video Stream" AvStreams. This hierarchical containment relationship would continue to a point where there were four AvSourcePorts that represent all of the primitive AvStreams, and thus all of the actual "produced" AvSignals. Again, viewed from any level, an AvSourcePort's aggregation structure parallels the AvStream's aggregation that is produced from that AvSourcePort.

Definition: ***Primitive*** **AvSourcePort**: a primitive AvSourcePort is one which produces a primitive AvStream. Primitive AvSourcePorts actually have AvPrimitiveCircuits (see below -- but think wire for now) attached, into which the corresponding AvSignals are injected when active.

Definition: ***Complete*** **AvSourcePort**: A complete AvSourcePort is one which is not contained by another AvSourcePort.

An AvSourcePort can be in either a "connected" or "disconnected" state. While complete AvSourcePorts are in the connected state, there will be one or more (multi-cast) associated *AvVirtualCircuits* into which it injects its complete AvStream. See the AvVirtualCircuit definition for more information.

It is intended that the AvSourcePort abstraction model physical source port hardware as well as "software" source ports of AvSignals such as software compressors and filters.

Audio/Video Sink Port

The ***AvSinkPort*** object class represents the termination point of a single AvStream. The aggregation hierarchy of an AvSinkPort corresponds to that of the AvStream which it sinks. While the aggregation structures correspond, not all of the component AvSignals may be accessible to, or used by, the AvSinkPort. Using the example of a combined VCR AvStream that has separate audio and video sub-AvStreams: an AvSinkPort on a video monitor would not utilize the audio AvStream even though it is logically available.

Definition: ***Primitive*** **AvSinkPort**: a primitive AvSinkPort is one which terminates a primitive AvStream. From a physical point of view, a primitive AvSinkPort actually has an AvPrimitiveCircuit attached, from which the corresponding AvSignal can be obtained.

Definition: ***Complete*** **AvSinkPort**: A complete AvSinkPort is one which is not contained by another AvSinkPort.

An AvSinkPort can be in either a "connected" or "disconnected" state. While complete AvSinkPorts are in the connected state, there is an associated *AvVirtualCircuit* out of which it obtains its (single) complete AvStream.

It is intended that the AvSinkPort abstraction model physical sink port hardware as well as "soft" sink ports of AvSignals for software processes.

Input Switch Ports

The ***AvInputSwitchPort*** object class derives its base behavior from the AvSinkPort class and extends that behavior by allowing for one or more "output" AvPrimitiveCircuits which represents the internal switch fabric paths to the various output ports.

Output Switch Ports

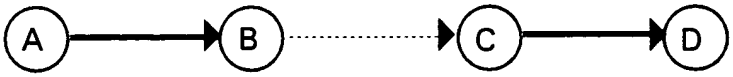
The ***AvOutputSwitchPort*** object class derives its base behavior from the AvSourcePort class and extends that behavior by allowing for an "input" AvPrimitiveCircuit which represents the internal switch fabric path to the corresponding AvInputSwitchPort.

Audio/Video Ports in general

The ***AvPort*** object class is an abstract class that represents all types of ports. In fact, the AvPort class is the parent class of both the AvSourcePort and AvSinkPorts.

Audio/Video Primitive Circuit

An ***AvPrimitiveCircuit*** object represents a real world point-to-point, static or dynamic, connection over which AvSignals (primitive AvStreams) travel. An AvPrimitiveCircuit may contain other AvPrimitiveCircuits in a series fashion so as to effect the higher level AvPrimitiveCircuit. It is intended that dynamic connections of either a physical or "soft" nature can be modeled as AvPrimitiveCircuits. Further, the HCS AV network model chosen is one in which a switch fabric is given the responsibility to "produce" an AvPrimitiveCircuit dynamically when requested to do so. After all, once an AvPrimitiveCircuit is created, either through hardwiring or dynamically, through switching, routing, etc., it behaves in the same fashion as any other AvPrimitiveCircuit.



Example: AvPrimitiveCircuit AD is composed of AvPrimitiveCircuits AB, BC and CD. AB and CD are static AvPrimitiveCircuits. They correspond to hardwired cable connections that are attached to primitive AvPorts. BC is a dynamically created AvPrimitiveCircuit within a switch fabric. NOTE: Architecturally AB and CD can either be simple direct point-to-point connections, or may contain dynamic AvPrimitiveCircuits in the same form of circuit AD. The complexity of an AvPrimitiveCircuit depends on the complexity of the hardware configuration which it is modeling. By representing the circuit interconnections in this hierarchical fashion, a wide range of hardware architectures can be accommodated.

Notice that AvPrimitiveCircuits are always directional and bound on one end by one kind of an AvPort and on the other by the other kind of an AvPort.

The intent is that the AvPrimitiveCircuit object class be able to represent both hardware and software A/V transports.

Audio/Video Switch Fabric

An ***AvSwitch*** object represents the complete AV switch fabric within an HCS system. The architectural intent is that all switching devices (hardware and software) be combined into a single instance of an AvSwitch. This includes both mechanical and software (or both) implemented solutions. This simplifies the usage of the switch fabric by other AV components. The abstraction presented by an AvSwitch is as a two-dimensional sparse matrix, with inputs in one dimension and outputs in the other.

When requested to do so, an AvSwitch will attempt to create a dynamic AvPrimitiveCircuit between the specified AvInputSwitchPort and AvOutputSwitchPort. This may not always be possible. The assumption is that there can be multiple outputs connected to one input (multi-cast).

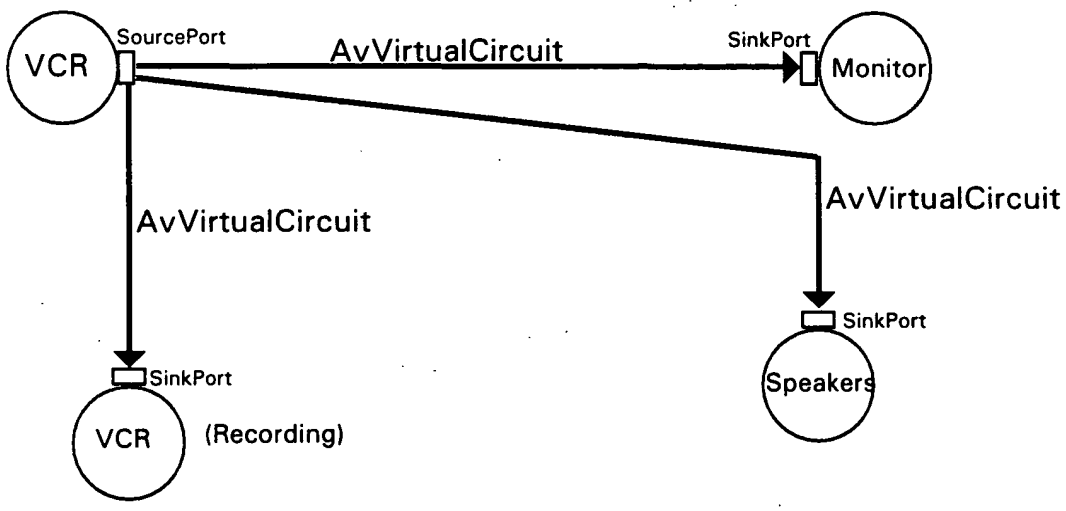
Audio/Video Virtual Circuit

An ***AvVirtualCircuit*** represents a temporal path, in terms of a set (1 or more) of parallel AvPrimitiveCircuits, over which an AvStream instance is carried. More specifically, an AvVirtualCircuit represents the connection between one complete AvSourcePort and exactly one complete AvSinkPort with the purpose of delivering the complete AvStream being produced. Keep in mind that multiple AvVirtualCircuits may be required just to route all of the desired component AvSignals of an AvStream to their destinations. There may truly be separate destination devices (e.g. Audio vs. Video). In other words, just because a single composite AvStream is produced from one AvSourcePort does not mean that all of its component AvSignals are destined to the same output device, and thus AvSinkPort.

An AvStream is multi-casted to more than one AvSinkPort via multiple AvVirtualCircuits.

Because of the hierarchical nature of AvPrimitiveCircuits, AvVirtualCircuits represent, and are associated with, all interconnects over which the actual AvSignals are routed. In other words, in terms of AvPrimitiveCircuits, an AvVirtualCircuit represents the two dimensional array of AvPrimitiveCircuits required to carry an AvStream (or part of it) from one AvSourcePort to one AvSinkPort.

Using the RGB AV Stream above as an example:



Audio/Video Program Players and Recorders

Objects that are of the *AvProgramPlayerRecorder* class represent the "devices", both physical and soft, that are the source of all AvSignals, and thus AvStreams within the HCS AV subsystem. Further, some types of AvProgramPlayerRecorders are capable of storing the content of AvStreams (AvSignals, etc...) for later access. AvProgramPlayerRecorders aggregate AvSource/SinkPorts which allow for inclusion of these devices into the AV Network. From a logical perspective, AvProgramPlayerRecorders either interpret the contents of *AvPrograms*, and/or produce AvPrograms when "recording". Architecturally, AvProgramPlayerRecorders can be arbitrarily complex in their implementation and need not represent physical hardware.

One of the important aspects of AvProgramPlayerRecorders is that they know what types of AvPrograms they can interpret.

Some types of AvProgramPlayerRecorders always deal with a fixed AvProgram, while others can be very dynamic (like those in JukeBoxes).

Audio/Video Programming

An ***AvProgram*** object instance represents a specific manageable unit of AV information that may be interpreted and presented (in the form of a complete AvStream) by an appropriate type of AvProgramPlayerRecorder. In general, AvPrograms contain all of the information needed to reproduce a complete AvStream similar to one that could have been used to originally create that AvProgram. Architecturally, AvPrograms can range from the very simple to the very complex. If fact, AvPrograms may contain other AvPrograms (even arbitrarily nested). In other words, things as simple as a live radio station broadcast or as complex as stored multimedia presentations can be represented as AvPrograms.

All AvPrograms contain two different types of attributes: descriptive and content. The methods used to represent and access AvProgram content are specific to the various kinds of AvPrograms and as such are not discussed further in this document.

NOTE: The AvProgram abstraction, as with most other aspects of HCS, makes heavy use of ASN.1 Object Ids for the purpose of hierarchical classification.

The following architectural attributes are common to all types of AvPrograms, and as such, define the base behavior for all AvProgram implementations. These are: *programDescription*, *programType*, and *programLocation*.

Attribute: ***programDescription***: the programDescription is a short, single line, textual description of the program.

Attribute: ***programType***: the programType attribute is an ASN.1 Object Id, with a preamble of 1.11, that classifies a program to a point that all other attributes, including content, can be interpreted; assuming specific knowledge of that type of program. The basic programType name-space is structured as:

1.11.1:	Broadcast program
1.11.2.1:	Stored program: Single access physical media (e.g. CD, VHS TAPE...)
1.11.2.2:	Stored program: Multiple access media (e.g. JPEG image...)

These are the only valid programType value preambles. AvPrograms that are of type 1.11.2.1 can only be used in one AvProgramPlayerRecorder at a time, while AvPrograms of type 1.11.2.2 can be played in multiple AvProgramPlayerRecorders at the same time.

Attribute: ***programLocation***: The programLocation attribute is an ASN.1. Object Id, with a preamble of 1.6.1, that identifies the exact location of the AvProgram. The actual meaning of "location" is dependent on the specific type of AvProgram. This does not mean that there are not architectural requirements placed on this attribute however. The basic programLocation name-space is structured as:

1.6.1.1:	AvProgram location: AvProgramPlayerRecorder associated.
1.6.1.2:	AvProgram location: AvProgramPlayerRecorder independent.

These are the only valid programLocation value preambles. It is very important to understand that these preambles classify different types of locations, not programs. In other words, the same type of AvProgram (e.g. VHS tape) could be in a AvProgramPlayerRecorder associated location (like a jukebox), or it may be AvProgramPlayerRecorder independent (like on a shelf). The reason for this architectural distinction between these different kinds of locations is that AvPrograms that are AvProgramPlayerRecorder-associated know how to load themselves. A good example of this is a CD that is in a jukebox. Such an AvProgram knows what jukebox it is in and can instruct that jukebox to load it into an appropriate AvProgramPlayerRecorder (drive).

Audio/Video Programming Pools

Instances of the ***AvProgramPool*** object class represent an organized collection of AvPrograms. AvProgramPools provide for the addition, removal, searching, browsing, and indirectly the presentation of their contained AvPrograms. Another aspect of AvProgramPools is that if they can contain (via reference) other AvProgramPools, this "containment" relationship is implemented in such a way as to make the access of sub-AvProgramPools transparent. This allows for the "gluing" together of multiple sub-pools while providing the illusion of a single AvProgramPool.

Programatically and architecturally, AvProgramPools provide for shared access and management in a networked environment. To effect this, other programatically-important helper-objects are defined as part of the AvProgramPool object family (like access-cursors and such).

Physical entities such as JukeBoxes and Image Bases will be represented by AvProgramPools.

AV Spatial Service

The **HcsAVSpatialService** object class is an abstract class from which spatially and AV oriented classes inherent their behavior. This currently is just a place holder in the classification hierarchy.

Audio/Video EntertainmentCenter

The term "entertainment center" is used because it provides a good intuitive "feel" for the behavior of this kind of object. The feel the user should get from an ***HcsEntertainmentCenter*** is as a collection-, or focal-point where AV sources can be selected and routed to outputs such as speakers, displays, and recorders.

HcsEntertainmentCenters know what their primary A/V outputs are and can automatically present user-selected AvProgramming to these via a "best fit" algorithm. In addition to this automatic mode of operation, HcsEntertainmentCenters also allow complex routing to occur; even outside their natural domain (space). The basic HcsEntertainmentCenter UI design approach is: "simple things are simple to do, but complex things are possible."

One goal is that there only be one kind of HcsEntertainmentCenter implementation.

From a programmatic point of view, the HcsEntertainmentCenter class orchestrates the collection of details that make up a user's request -- keeping in mind that a user can change their mind, and do things in an arbitrary order. The important thing to understand is that this "collecting of user intent" is done at a very abstract level.

HcsEntertainmentCenters provide a UI-framework through which other AV objects present their UI components. For example, a AvProgramPlayerRecorder would present itself, and interact with the user, through HcsEntertainmentCenters. Also keep in mind that HcsEntertainmentCenters themselves present their UIs ultimately through different kinds of HcsUserControlPoints and as such tailor their behavior to these different environments.

Back to the Functionality Model

At this point the reader is encouraged to review object model definitions as there are subtle inter-relationships between the classes in this object model.

Let's now take a look at the AV Functionality Model and map the various object classes to the layer of functionality which they implement.

Layer	Objects
Service	HcsAvSpatialServices, HcsAvCollections, and HcsEntertainmentCenters
Management	AvSignals, AvStreams, AvPrograms, AvProgramPools, and AvProgramPlayerRecorders.
Network	AvVirtualCircuits , AvPorts, AvSwitch, and AvPrimitiveCircuits

Each of these layers are discussed below, starting from the Network layer and working up to the Service layer. It is important to understand that the layering indicated by this model does not imply that the lower level kinds of objects (like at the network level) do not know about higher level objects -- this is not a layered communications reference model.

Network Layer

The objects that implement the Network layer of functionality represent the "plumbing" of the AV system. Specifically, the sources, circuits (dynamic and static), routes, and destinations of all AV information. Another way to think about the objects at this level is in terms of information that they contain. At any given time the objects at this level represent a schematic of the interconnections between all AV components in an HCS installation.

The document entitled "HCS AV Network Components Design" defines each object class of this layer in detail.

Management Layer

The objects that implement the Management Layer of functionality are responsible for managing all devices, processes, and information from an AV viewpoint. This implies device "drivers" in the form of AvProgramPlayerRecorders, and live and stored information through the AvProgram, AvSignal, AvStreams, and AvProgramPool implementations.

This is the layer that does most of the work in terms of actively carrying out user requests.

See the document entitled "HCS AV Management Components Design" for a complete definition of the objects at this level.

Service Layer

The objects that implement this layer of functionality are responsible for mapping various Management Level components into a natural UI framework for the user. Typically a space-centric approach is used like that of the HcsEntertainmentCenter. If other, nonspace-centric, paradigms are needed, other types of components could be introduced at this level without affecting components at the other levels.

See the document entitled "HCS AV Service Components Design" for a complete definition of the objects at this level.